Scheduling a Kubernetes Federation with Admiralty

Igor Sfiligoi – UC San Diego
on behalf of the PRP/Nautilus group

UC San Diego

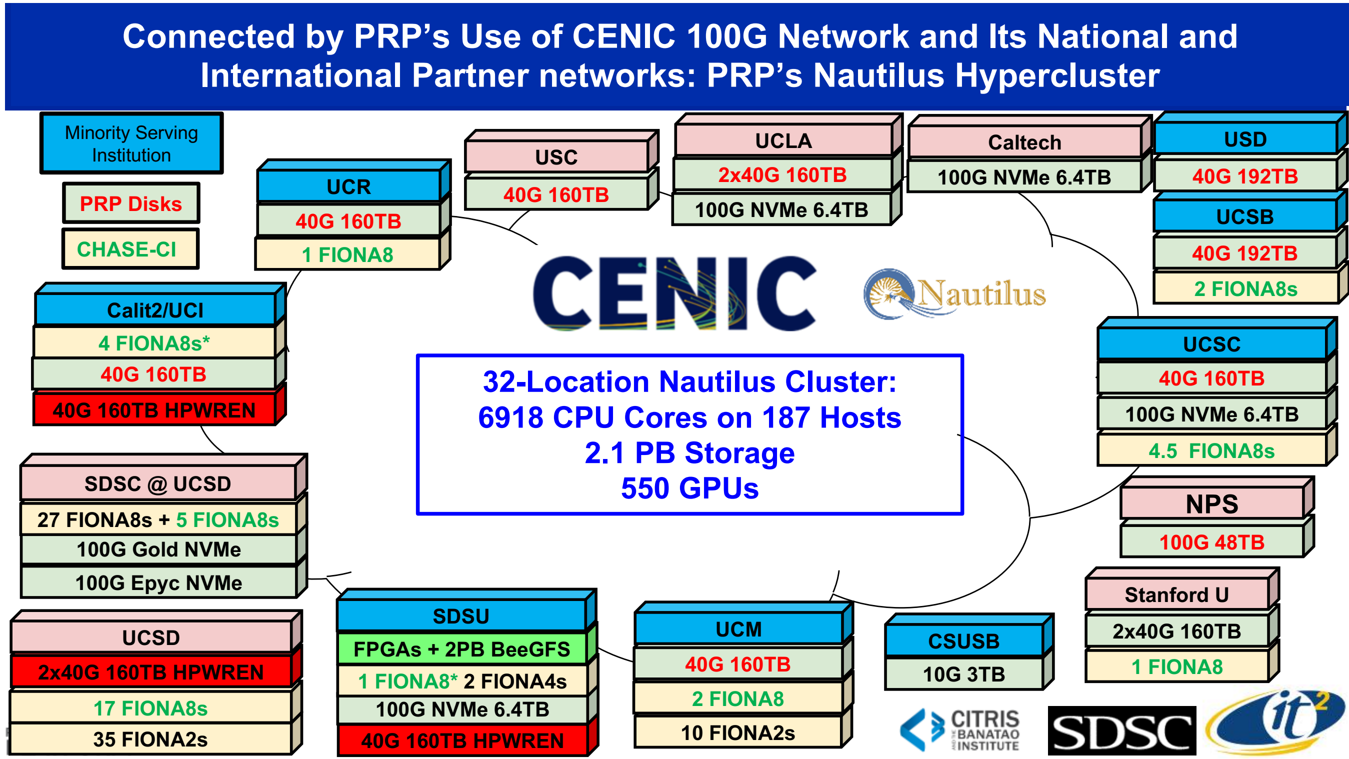
SDSC
SAN DIEGO SUPERCOMPUTER CENTER

it²

PRP PACIFIC RESEARCH PLATFORM

PRP, Nautilus and Kubernetes

- The Pacific Research Platform (PRP) has been using Kubernetes since 2016
  - Started as a way to conveniently schedule network test services
  - Evolved in being a convenient platform for ML research
- OSG has had a CE gathering opportunistic cycles for over a year now
  - As well as orchestrating some of its services e.g. StashCache and Frontends



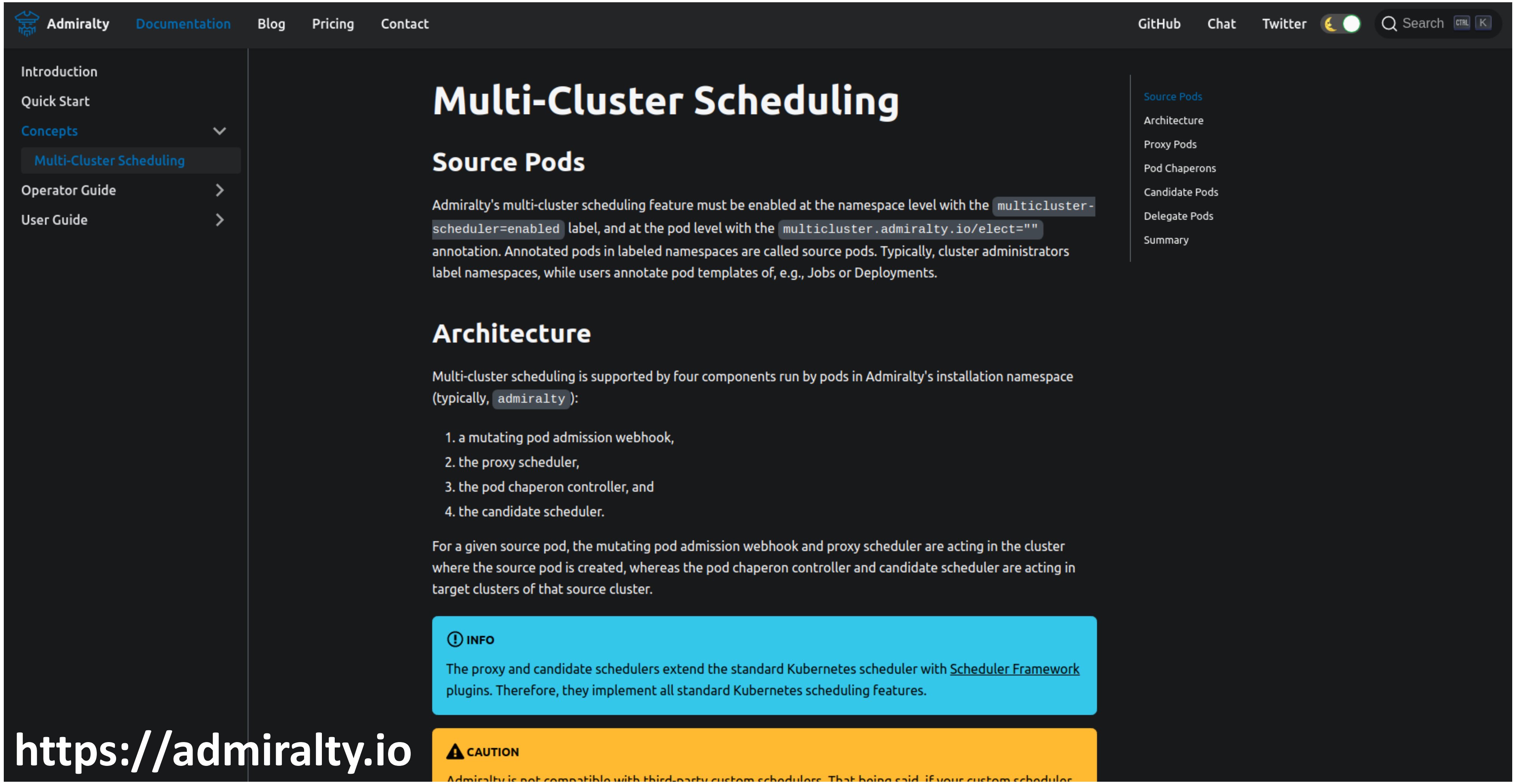
Why federation?

- PRP/Nautilus has been steadily growing
  - It now has nodes also in Asia, Europe and Australia
  - While successful, we do understand not everyone will want to join the club
- Separate administration domains
  - We even have the use case at UCSD
  - PRP Nautilus and SDSC Expanse will operate separately, but will work together through federation
- Multiple platforms
  - PRP has an IoT component, where ARM CPUs rule
  - Having a dedicated ARM k3s and federating with it ended being simpler

Driving principles

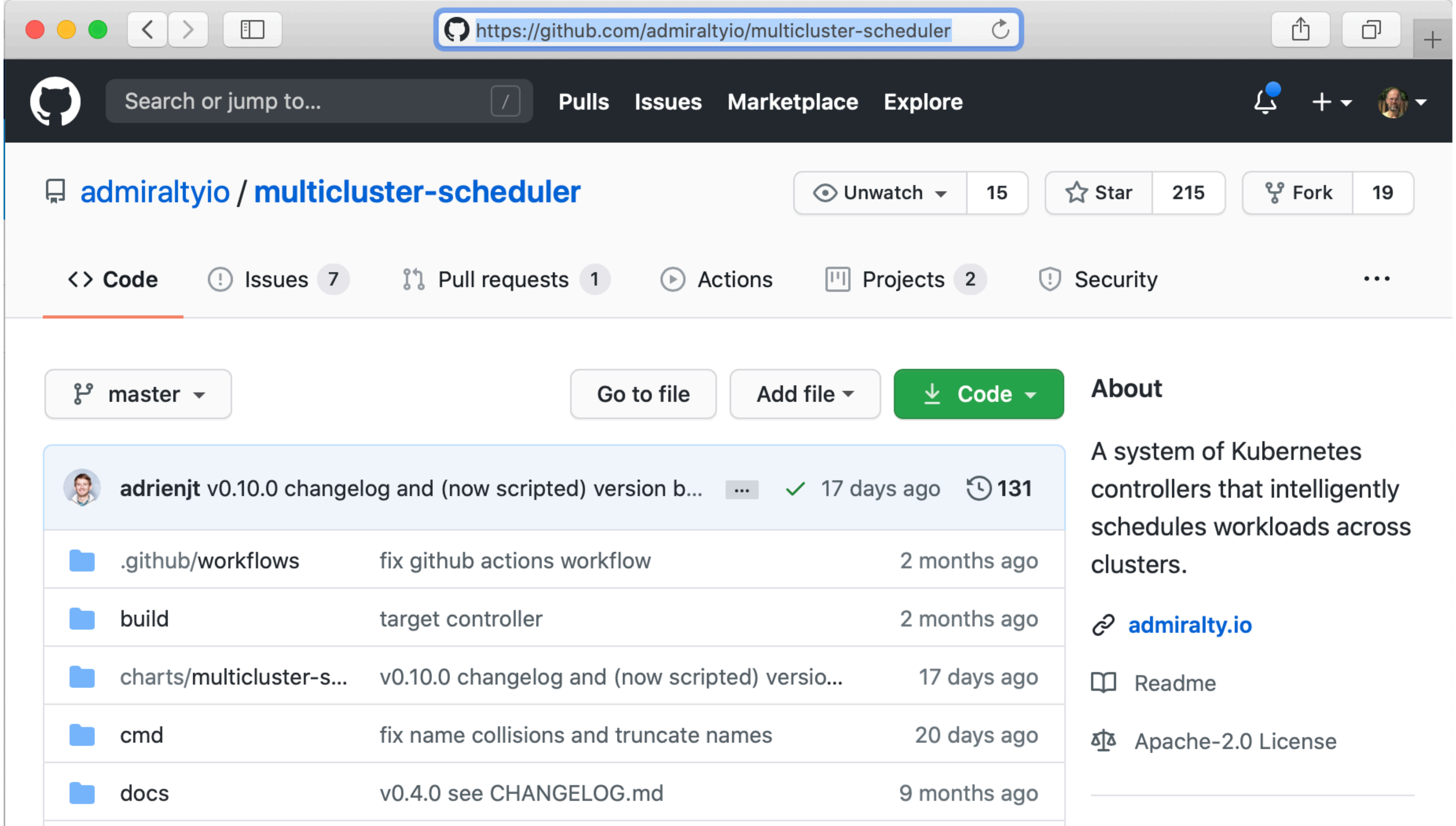
- We wanted a “native Kubernetes” solution
  - I.e. kubectl should be all that the user needs
- We did **not** want a centralized solution
  - All participating Kubernetes clusters should be on equal playing field
  - Each Kubernetes cluster should be able to participate in any number of federations
- We did not want to do any development ourselves
  - Helping with testing OK
  - Occasional patch OK
  - But no long-term maintenance

Admiralty’s Multicluster-Scheduler



https://admiralty.io

Admiralty's Multicluster-Scheduler



Admiralty on Nautilus

- Currently running 0.10.0-rc1
- Have been federating with
  - ARM-based k3s
  - PacificWave Kubernetes cluster
  - Google Cloud Kubernetes cluster
  - Kubernetes Cluster inside Azure
- Getting ready to federate with
  - Expanse’s Kubernetes partition
  - A Windows-based Kubernetes cluster

Installing Admiralty

- Pretty well documented in github: https://github.com/admiraltyio/multicluster-scheduler/tree/v0.10.0-rc.1
- Source and target cluster both need Admiralty installed
  ```
  helm install cert-manager …
  helm install multicluster-scheduler admiralty/multicluster-scheduler …
  ```
- Create secret in target cluster and propagate to source cluster
  (targer) `kubemcsa export -n klum c1 --as c2 >s.yaml`
  (source) `kubectl -n admiralty apply -f s.yaml`
- Whitelist target cluster in source cluster *(helm update …)*
- You are pretty much good to go!
  - Pods in source cluster just need to add an annotation
    ```
    metadata:
      annotations:
        multicluster.admiralty.io/elect: ""
    ```

Installing Admiralty

- Admiralty creates a set of new resource types

```
Igors-MacBook-Pro:~ isfiligoi$ kubectl api-resources |grep admiralty
clustersources                   csrc               multicluster.admiralty.io        false        ClusterSource
clustersummaries                 mcsum              multicluster.admiralty.io        false        ClusterSummary
clustertargets                   ctg                multicluster.admiralty.io        false        ClusterTarget
podchaperons                     chap               multicluster.admiralty.io        true         PodChaperon
sources                          src                multicluster.admiralty.io        true         Source
targets                          tg                 multicluster.admiralty.io        true         Target
```

- Target clusters can be seen as virtual nodes

```
Igors-MacBook-Pro:~ isfiligoi$ kubectl get nodes |grep admiralty
admiralty-igor-gke-us-central                      Ready                              cluster    7d18h
admiralty-k3s                                      Ready                              cluster    7d18h
admiralty-nautilus                                 Ready                              cluster    7d18h
```

Installing Admiralty

- We have been mostly using one-way federation
  - Nautilus as source, others as targets
- Nautilus can easily be the target, too
  - Admiralty allows for arbitrary mesh
  - Federation with SDSC Expanse is expected to be both ways

Scheduling to target clusters

- Admiralty's Multicluster-Scheduler is a real Kubernetes scheduler
  - Users do not get to pick explicitly the target
  - Offload happens based on standard requirements and preferences
  - Users just have to opt-in
- When there are nodes in multiple possible clusters that match
  - Admiralty will consider only clusters that have free matching nodes
  - Which target cluster will be picked is (mostly) non-deterministic
  - If no target clusters have any available matching nodes,
    the pod remains pending in the source cluster (only)
- Priorities and preemption work as you would expect them to

Scheduling to target clusters

Under the hood, uses the standard k8s filtering and scoring mechanisms

Node selection in kube-scheduler

kube-scheduler selects a node for the pod in a 2-step operation:

1. Filtering
2. Scoring

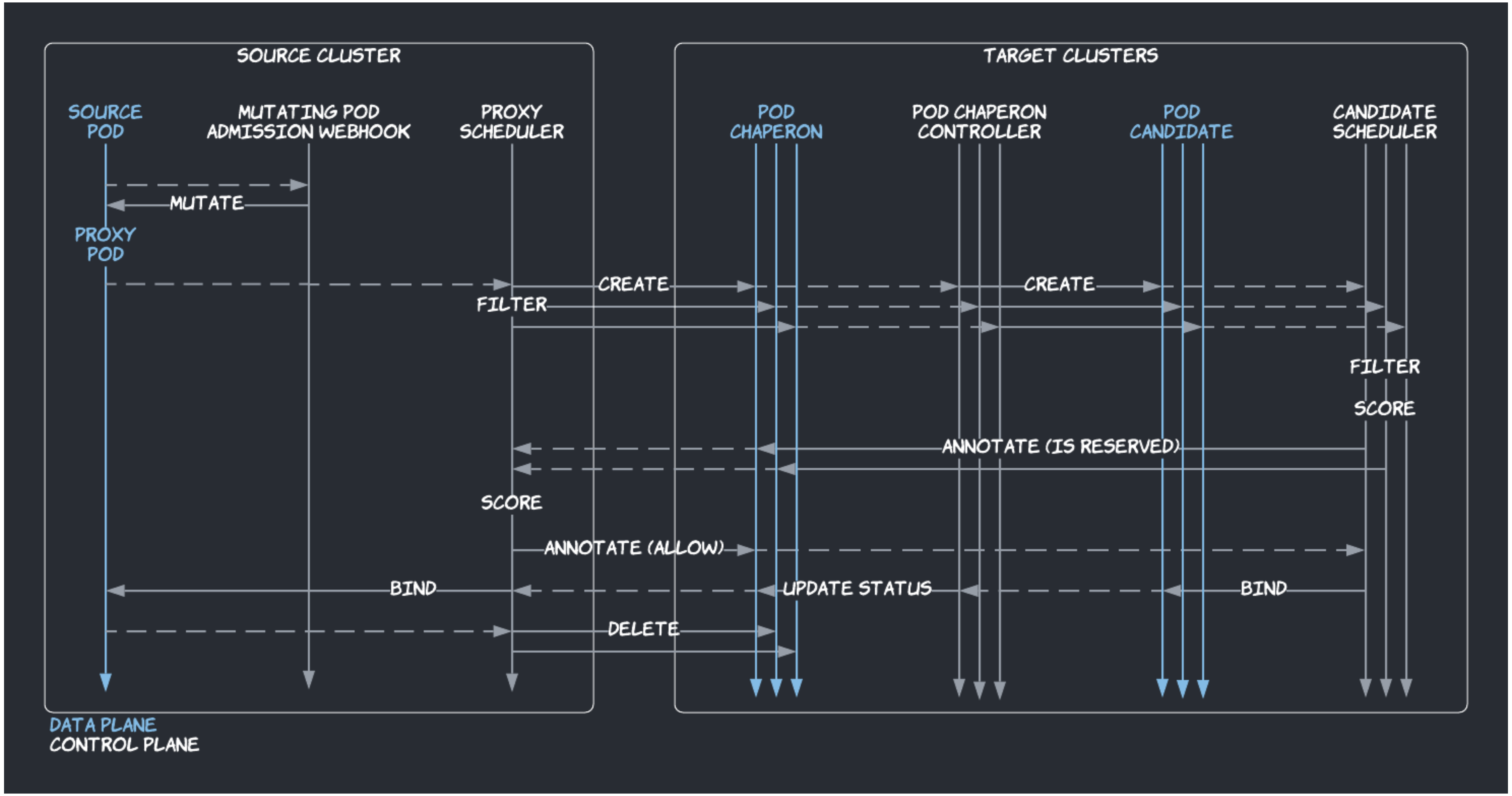
The *filtering* step finds the set of Nodes where it's feasible to schedule the Pod. For example, the PodFitsResources filter checks whether a candidate Node has enough available resource to meet a Pod's specific resource requests. After this step, the node list contains any suitable Nodes; often, there will be more than one. If the list is empty, that Pod isn't (yet) schedulable.

In the *scoring* step, the scheduler ranks the remaining nodes to choose the most suitable Pod placement. The scheduler assigns a score to each Node that survived filtering, basing this score on the active scoring rules.

Finally, kube-scheduler assigns the Pod to the Node with the highest ranking. If there is more than one node with equal scores, kube-scheduler selects one of these at random.

https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/#kube-scheduler-implementation

Scheduling to target clusters



Other features

- Admiralty has several other features we have not explored yet
- Three potentially interesting options:
  - Multi-cluster services, using a
    load-balancing across a Cilium cluster mesh
  - Identity federation (instead of shared secrets)
  - Federation with Targets lacking a public IP (reversed connectivity)

Conclusion

- Admiralty has been in use in the PRP k8s cluster/Nautilus for some time now
- Works as advertised for our main use cases
- We are planning to use it to expand to more clusters in the future

Acknowledgments

- This work was partially funded by the US National Science Foundation (NSF) under grants OAC-1826967, OAC-1541349, MPS-1148698 and OAC-1841530.

PACIFIC RESEARCH PLATFORM